IoT Resiliency through Edge-located Container-based Virtualization and SDN

Nikki John B. Florita
University of the Philippines Diliman
Quezon City, Philippines
nbflorita@up.edu.ph

Miguel Luis R. Sesdoyro
University of the Philippines Diliman
Quezon City, Philippines
miguel_luis.sesdoyro@upd.edu.ph

Julian Troy C. Valdez
University of the Philippines Diliman
Quezon City, Philippines
jcvaldez1@up.edu.ph

Richard F. Guinto
Samsung Research Philippines
Taguig City, Philippines
rf.guinto@samsung.com

Wilson M. Tan
University of the Philippines Diliman
Quezon City, Philippines
wmtan@dcs.upd.edu.ph

2021 IEEE Asia Pacific Conference on Wireless and Mobile (APWiMob) | 978-1-7281-9475-2/21/$31.00 ©2021 IEEE | DOI: 10.1109/APWIMOB51111.2021.9435269

***Abstract*—Despite the falling cost of electronics, the adoption IoT systems in households ("smart homes") is hampered by their strict reliance on stable and reliable Internet connection, which is not readily available in all places around the world. This reliance is greater still in sensor-actuator setups that are comprised of products from several manufacturers or ecosystems, since the packets have to traverse and be processed by different cloud-based servers. In this paper, we propose a *resilient IoT system* which enables household IoT systems to remain functional even in the absence of an Internet connection. The system utilizes container-based virtualization in locally hosting cloud server functionality and software-defined networking (SDN) in hiding the packet redirection and providing a seamless network experience for the IoT devices. The system was built and its performance profiled as it utilized different hardware for its container-based virtualization host.**

I. Introduction

Today's smart homes have emerged from the integration of the Internet-of-Things (IoT) paradigm with cloud computing [1]. The typical architecture consists of a local network of home IoT devices (sensors, cameras, etc.) that send data directly or indirectly (through a gateway device) to the cloud. The cloud server performs data analysis and control computations, and sends the results back for execution [2].

The key advantage of leveraging the cloud for smart home services is the on-demand availability of resources for storage and computations. However, the heavy reliance to it poses challenges in ensuring system operation during cloud disconnection. Several works have proposed cloud disconnection-tolerant schemes using alternative data transmission. In [3], smartphones are envisioned as generic gateways from IoT peripherals to the cloud. The architecture allows data from Bluetooth Low Energy (BLE)-enabled peripherals to hop along common BLE-enabled smartphones to communicate to the cloud. The idea is to leverage on smartphones' near-constant internet connection. While the approach is promising due to the ubiquity of smartphones, it requires specific communication protocols limited to a set of peripherals. In [4], a publish/subscribe data transmission architecture was implemented for replicating and buffering data. This ensures continuous availability of data at the edge, and resumes normal operation when network outage has been resolved.

Recently, fog computing was introduced as a platform between end devices and the Cloud, typically located at the edge of the network [5], hence also called edge computing [6]. The platform enables new applications for IoT, such as offloading some computations from cloud to edge to improve response time [7], data hub for local processing [8], among others. In [9], its application for network-constrained environments is proposed. Specifically, the feasibility of Raspberry Pis (rPis) as edge devices at which to migrate Dockerized cloud applications is evaluated. In [2], the Resilient Home-Hub (RES-Hub), a dedicated edge device, was proposed to take over the required services when the cloud is unavailable. The system runs in resilient mode and performs smart home commands based on predefined emergency and user specifications.

In this study, we adopt the concept of RES-hub [2], employing an edge device that operates during cloud disconnection. In contrast to predefined functionalities, we explore the capability of our hub to run Dockerized servers that aim to mimic the IoT cloud(s), as well as providing a more manufacturer agnostic approach for resiliency, and a slight degree of decentralization for resiliency features. The rest of the paper is structured as follows. In Section II we discuss the basic IoT system architecture. In Section III we detail the proposed resilient IoT system and its components. In Section IV we detail the rerouting process for packets during the operation of the resilient IoT system; and in Section V we show the performance metrics of the resilient system across a variety of hardware. In Section VI, we summarize the results of our performance metrics and give our conclusion.

II. The System

An example of typical IoT architecture can be seen in Figure 1. It consists of IoT devices connected to an access point that allows it to communicate with services throughout the cloud. Section II-A will cover the basic components of the system, while Section II-B will discuss normal system operation.

A. System Components

In order to replicate the setup from Figure 1 to be used in testing at Section V, the set of characteristics and interre-

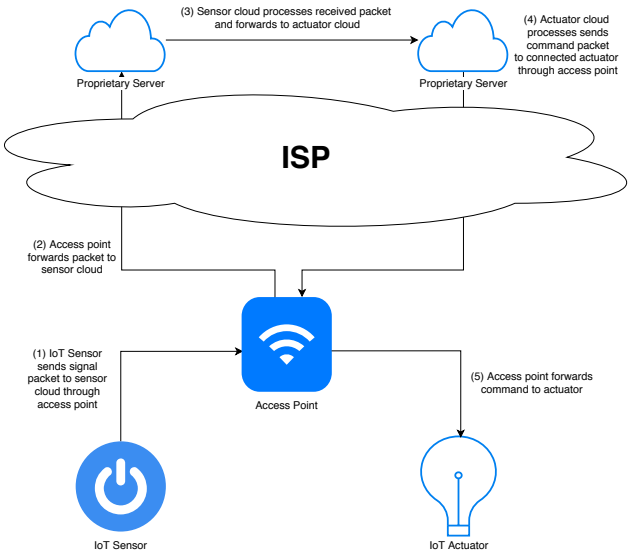


Fig. 1: Normal IoT Architecture

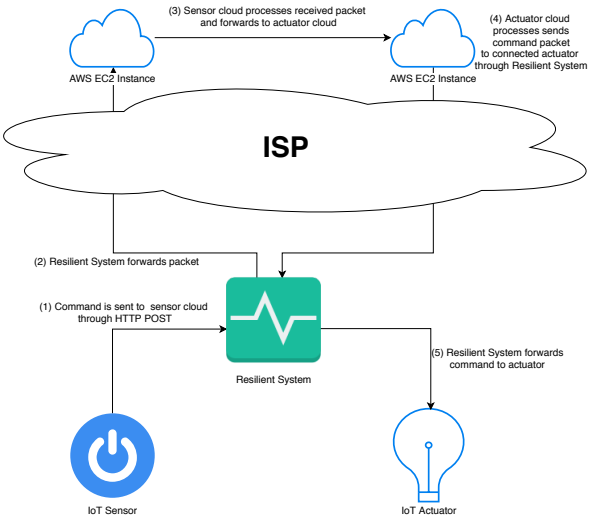


Fig. 2: Resilient IoT System

lationship of the components need to be well defined within the context of experimentation.

1) IoT Sensor: The IoT sensor is any host that is capable accepting user input, and sending HTTP requests to its designated cloud service; process (1) in Figure 1.

2) Iot Actuator: The IoT actuator is any host that is capable of initiating websocket connections to its designated cloud service. Through this websocket connection the IoT actuator is able to receive instructions from the cloud service. In Figure 1, (4) shows that once the cloud service sends an instruction packet, it is received by the IoT actuator in (5).

3) Access point: The access point is a router or gateway from which the IoT devices can connect to an ISP to.

4) Cloud Service: The cloud service component of the system is designed to emulate a standard IoT cloud setup, as seen in Figure 1. Each device is presumed to be from a different vendor with a proprietary cloud. All communication between IoT devices is facilitated by their respective cloud server to better represent the proprietary nature of each device.

Each cloud server uses two methods of communication, HTTP REST and websockets. Input devices such as input from the IoT sensor, server configuration, and server-to-server communication use REST. Output devices such as the IoT actuators communicate with the server through websockets. REST provides easy to setup TCP/HTTP communication. Websockets are used to immediately send events to endpoint devices. Individual device timeouts are manually checked by the server.

B. Normal IoT Operation

Figure 1 shows the operation of the system. (1) shows the IoT sensor sending an HTTP request invoked upon user input to its designated cloud service. The access point then forwards this request to the cloud service (2). Once the cloud service processes the request sent by the actuator, it will perform lookups on its configurations to see what IoT actuator device the specific request received is supposed to affect, it then contacts the cloud service of the IoT actuator device to be triggered (3). (4) then shows the cloud service of the IoT actuator device sending an output packet containing a specific action to be executed based from the input it has taken from (3). (5) then shows the IoT actuator accepting the output from the cloud service as input and performs the instruction it comes with.

III. Resilient System

The proposed resilient system would function similarly to the system in Figure 1. Software Defined Networking (SDN) would be used, allowing for rerouting of traffic to Docker containers containing emulated versions of cloud servers. As shown in Figure 2, the resilient system's major difference is the replacement of the regular access point with an SDN setup. The SDN setup allows for redirection of traffic during periods where no internet access is available. There are two major components for the resilient system: the Dedicated Docker Host, and the Packet Rerouter, as seen in Figure 3 and 4 respectively.

A. Packet Rerouter

The packet rerouter is the system tasked with performing the **packet rerouting process** which would be explained more in detail in Section IV. The packet rerouter is comprised of multiple components, each with a distinct purpose, as seen in Figure 4.

1) SDN Capable Switch: The SDN Capable Switch enables the system to reroute the packets from cloud to virtualized server by matching and modifying packet headers, as defined by the flows given by the controller software. A Zodiac GX switch [10] with OpenVswitch capabilities is used for this implementation.

2) SDN Controller: The SDN Controller hosts the controller software which defines the flows sent down to the SDN Capable Switch. Any machine capable of running Python and the Ryu library may be the SDN Controller. In the following tests, an Intel NUC NUC6CAYS [11] is used.

978-1-7281-9475-2/21/$31.00 ©2021 IEEE

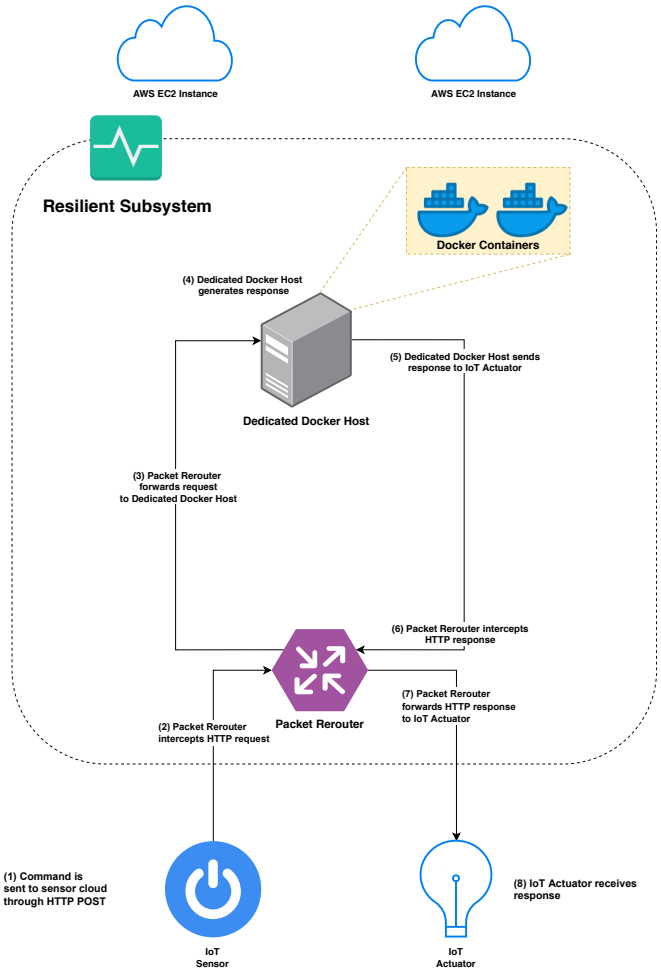


Fig. 3: Operation of Resilient Setup without Internet Connection

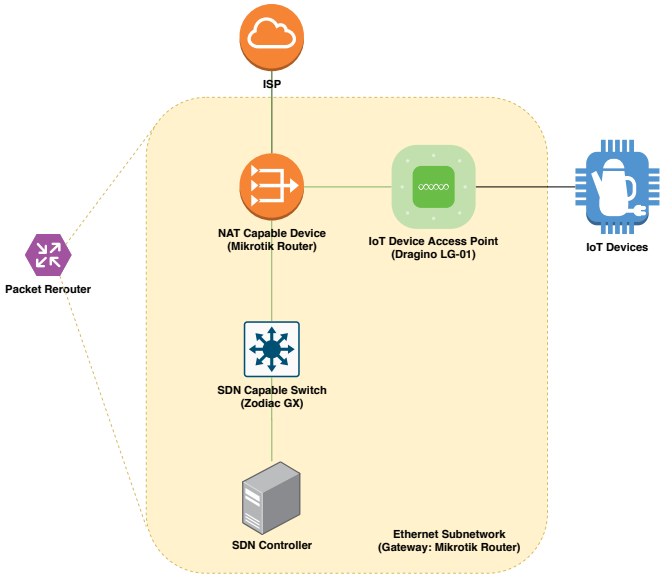


Fig. 4: Packet Rerouter

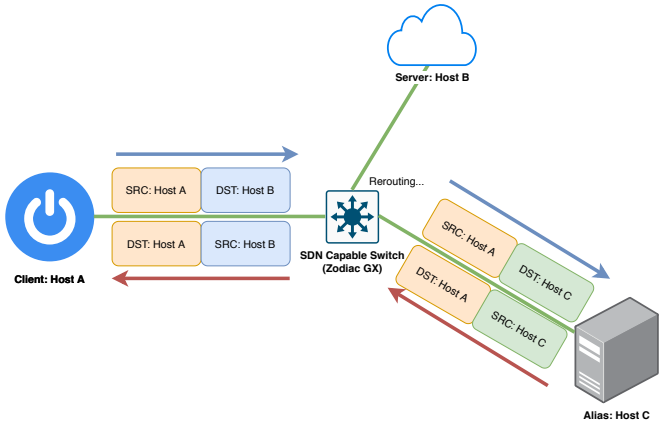


Fig. 5: Sample network topology employing the rerouting process

3) IoT Device Access Point: The IoT Device Access Point is the wireless access point to which the IoT devices connect to. This access point is used to aggregate all IoT devices into a single network entity via NAT, reducing the number of flows required for the system to function. A Dragino LG-01 [12] is used for this implementation.

4) NAT Capable Device: The NAT capable device serves as a gateway for the rest of the system, and to act as a sandbox for the Resilient System. It defines the subnetwork which the Resilient System is on, allowing packet rerouting to be restricted to the subnetwork alone. A Mikrotik hEX [13] is used for this purpose.

B. Dedicated Docker Host

The Dedicated Docker Host hosts the Docker containers used to mimic the functionalities of cloud servers. Each Docker container corresponds to a single IoT device manufacturer, though multiple containers for a single manufacturer may exist for the purposes of load balancing. The Docker containers directly use the IP address of the machine which they are hosted. To resolve port conflict issues, each Docker container TCP port is mapped to a different port on the host machine.

IV. Packet Rerouting Process

The packet rerouter aims to change the behavior of a normal client-server architecture where network packets sent by the client, which are dedicated for the corresponding server, and reroute them towards a host that is part of the same sub-network, known as the client host. An example of a high-level view of how this process works is shown in Figure 5. The client's network traffic which are all portrayed as initially being sent to the server host are then rerouted towards an *alias* host, the *alias* host then responds normally to the client host as if the network traffic was actually meant for itself and finally the packets are again modified in order for the received packets by the client host be seen as coming from the server host instead of the *alias* host.

From Figure 3, the packet rerouting process executed by the Resilient system upon loss of connection is shown. Similar to the normal operation in Figure 2, the IoT sensor first sends out an HTTP request to its dedicated cloud service, but once it reaches the resilient subsytem, instead of forwarding

978-1-7281-9475-2/21/$31.00 ©2021 IEEE

it to the ISP, it is instead intercepted by the packet rerouter (2) and forwarded to the Dedicated Docker host component (3). Once the Dedicated docker host receives this request, it then processes it and generates an output similar to what the genuine cloud services would normally respond with (4), the Dedicated docker host then sends a response for IoT actuator (5). Before the response of the Dedicated docker host reaches the IoT actuator, it is first intercepted by the Packet Rerouter component (6) and only then is it forwarded to the IoT actuator (7).

V. Performance Evaluation

There are two setups that the resilient subsystem was tested in: the **Variable IoT devices** setup and the **Load Balanced setup**. Both experiment setups execute the process defined in Section IV, with each setup having a different set of parameters to be tested: IoT device count for the **Variable IoT devices** setup, and containers for the **Load Balanced setup**.

To emulate a large number of IoT actuators and sensors, we utilized a host running Mininet [14] which emulated IoT devices using virtual network interfaces. The setup can be seen in Figure 3.

Both experiment setups use a Supermicro 118-14 Server [15] for emulating IoT device pairs using Mininet. An Intel NUC NUC6CAYS [11] and an Intel NUC Kit NUC8i7HVK [16] are both used as the Dedicated Docker Host each having an independent set of experiment runs. The Intel NUC NUC6CAYS would be referred to as the *Weak NUC* and the Intel NUC Kit NUC8i7HVK as the *Strong NUC* in succeeding sections due to their specification differences specifically in Memory and CPU, with the *Weak NUC* only having 8GB of RAM and an on-board Intel Celeron Processor J3455 to the *Strong NUC*'s 32 GB of RAM and on-board Intel Core i7-8809G processor.

A. Variable IoT devices

The following experiment is performed to test the system under variable amounts of IoT devices present. The experiment setup is shown in Figure 6, this setup is based off the general diagram of the resilient IoT system from Figure 2 and also performs the same operations, but instead of having physical IoT device components, a Supermicro 118-14 Server emulating the IoT device pairs is used.

Figure 7 shows the results of the test with varying amounts of IoT actuator and sensors. The results also show two different two different sets of values for the *Strong NUC* and *Weak NUC* which correspond to two experiment setups that used different hardware components as their Dedicated Docker host. There would only be a single pair of containers being run by the Dedicated Docker Host in each setup. One container is assigned for communicating with the IoT sensor Devices while the other is assigned for communicating with the IoT Actuator Devices.

Figure 7a shows the results of measuring the delay performance metric after performing the multiple experiments using sets of 2, 10, 30, 60, 100, 150 and 200 Mininet hosts each independently conducted. This would translate to 1, 5, 15, 30, 50, 75 and 100 emulated IoT sensor-actuator pairs respectively. Each IoT sensor sends total of 200 requests packets sent at a rate of 0.5 requests per second. The results show the average delay of all individual requests across all device pairs along with the corresponding standard deviation value.

As seen in Figure 7a the delay drastically increases from 5 IoT device pairs to 15 IoT device pairs from around 0.225 - 0.339 seconds to 18.146 - 19.125 seconds after which it slowly increases reaching up to 25.884 seconds at 100 device pairs. These results show that the performance of the resilient subsystem with only 1 running container pair starts to have poor performance serving from 10 total IoT devices to 30. Even though the Strong NUC proves to be slightly better in terms of delay performance responding 0.090 ms up to 2.059 seconds faster on average than the Weak NUC above 10 device pairs, it still goes past 20 seconds which is poor in terms of delay performance.

According to the standard deviation values shown per setup, most of the values are very far from the mean delay, with some standard dev to mean ratio reaching up to 93% on the 15 pair setup for the strong NUC. These standard deviation values show that as the number of IoT device pairs increase, the more disparate the delay readings become, with some readings nearing only 7% of the mean delay alongside delay readings of 193% size of the mean delay on the same experiment setup. The standard deviation values stabilize at around 17 seconds for setups above 30 device pairs due to the 60 second hard limit for packets before being marked as losses, which means any potential delay readings that could influence the mean to trend upwards even more are simply counted as losses instead.

Figure 7b shows the amount of request packets sent by the IoT sensors in which the actuator IoT device has not received a corresponding response, these are considered to be packet losses experienced by the system during the experiment runs. It can be seen that as the amount of emulated IoT devices increase, more packets are lost in transmission. For this experiment setup, any requests sent by the IoT sensor device that has not reached the corresponding IoT actuator device in 60 seconds are considered losses.

Similar to the behaviour shown in Figure 7a, The amount of losses start to increase drastically from 15 to 30 device pairs where it jumps from 10 - 13% to 60% packet loss eventually reaching 90% loss at 100 device pairs, with the Strong NUC only being at most 2.16% less losses than the Weak NUC.

Overall, the resilient subsystem with only 1 container pair serving all IoT device pairs does not scale well as the amount of active IoT device pairs increase, incurring 60% up to 90% losses and above 20 seconds delay. Therefore, an alternative setup which utilizes the dedicated docker host's ability to maintain multiple active containers can be leveraged by the SDN controller to support a load balanced scheme. This alternative setup is explained at Section V-B.

978-1-7281-9475-2/21/$31.00 ©2021 IEEE

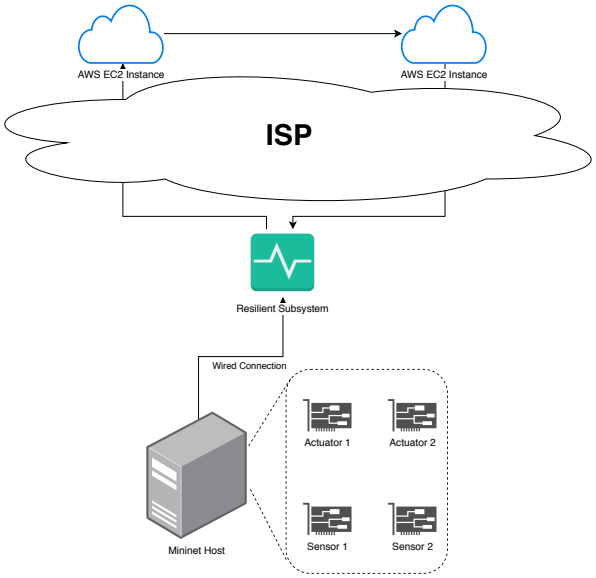
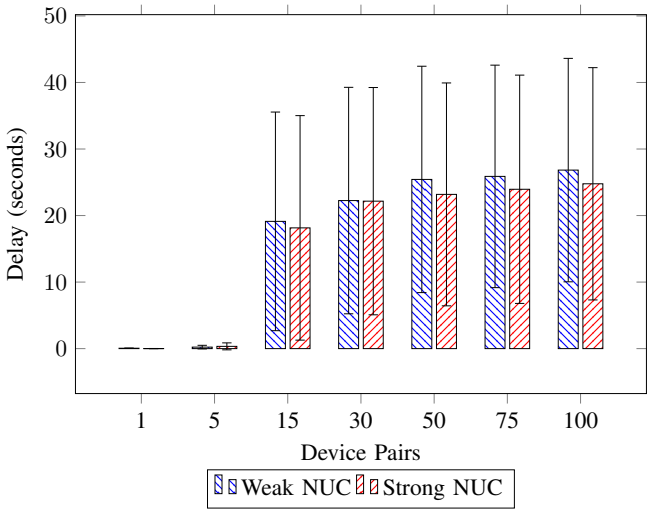
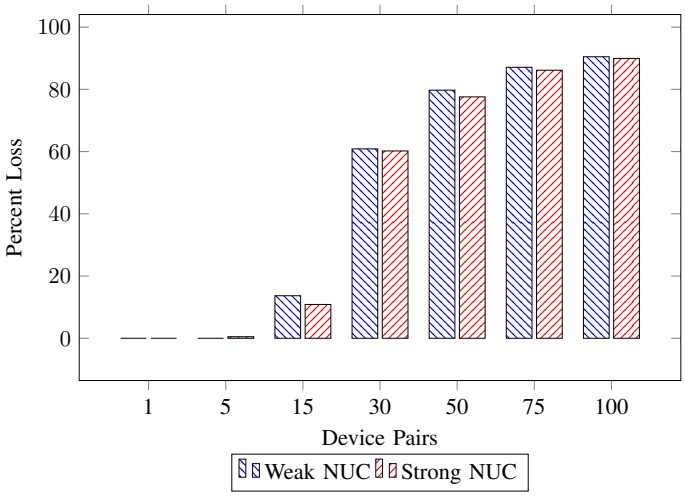


Fig. 6: Experiment setup with Mininet host



(a) Delays



(b) Losses

Fig. 7: Delay results, variable IoT devices

B. Load balancing

The performance of the setup described in Section V-A drops significantly even at low parameter values. Therefore an alternative configuration of the resilient subsystem with additional basic load balancing capabilities was implemented and tested.

In Figure 8, an experiment setup where a range of packets from a group of IoT devices is evenly serviced by a variable number of containers. *M* and *C* are constants that refer to the number of IoT device pairs and number of requests per IoT device pair respectively. These values are used to represent the total amount of requests that would be sent out from the IoT device group as $M*C$. With the total amount of requests $M * C$ and number of containers in the Dedicated Docker host N, the total number of requests to be sent per container pair could then be represented as $(M * C)/N$.

Figure 9 shows the difference in the measured performance metrics from a constant number of 75 IoT device pairs as well as a constant amount of 200 requests for each pair. The setup ran on a variable amount of containers that evenly serviced the workload. As seen from Figure 9a the average value of the delay performance metric for all the IoT device pairs improve as more containers are run to service the requests.

The first data point of 1 container pair is equivalent to the 75 IoT device pair setup from Figure 7a, but as the load balancing proper gets enabled starting from the 5 container pair setup, the performance improves immensely for both the Weak and Strong NUC setups. The delay measurements go from 25 seconds on the 1 container pair setup, to 19 seconds on the 5 pair setup to around 0.2 seconds on 15 container pairs and above. Although, the Weak NUC setup at 75 container pairs consistently ceases to function due to lack of memory at the beginning of the test, therefore its measurements were instead set to the maximum values (infinite delay, 100% loss).

Similarly, as shown in Figure 9b , the losses experienced by the IoT device pairs decrease as more containers are running to serve the requests. Losses go from 86% in the 1 container setup, to around 15% on 5 pairs into almost 0 in succeeding tests. However, similar to the delay measurements, the Weak NUC ceases to function at the 75 container pair setup and therefore losses are set to 100%.

The results have shown that the Strong NUC performs better than the Weak NUC overall but only by a small margin, and is also able to host more containers where the weak NUC normally stops responding as with 75 container pairs.

VI. Conclusions

In this paper we have proposed a resilient system, dependent on software-defined networking and container-based virtualization, which aims to ensure the reliability and functionality of IoT devices when internet access is inconsistent or slow. We implemented and tested our proof concept using virtualized IoT devices, and measured metrics such as packet loss between IoT devices, CPU usage within our IoT

978-1-7281-9475-2/21/$31.00 ©2021 IEEE

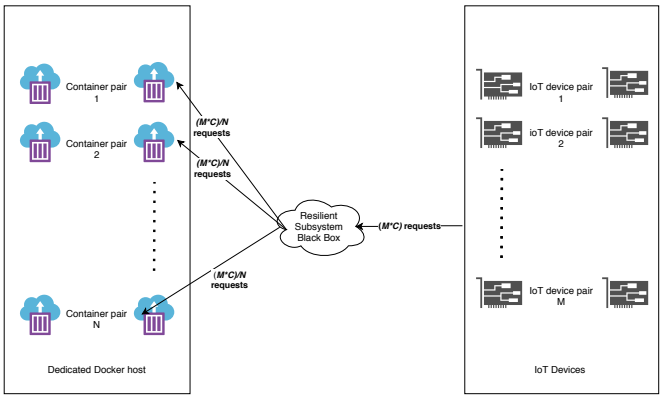


Fig. 8: Load balanced experiment setup

virtualization machine, and delay between IoT devices and containers.

As seen from the results in Section V, depending on how computationally intensive the cloud processes would be, hosts serving as the Dedicated Docker Host with more resources such as memory and CPU clock speed could potentially improve the performance of the resilient subsystem. It would be wise to choose a Dedicated Docker Host model of ample capacity that is able to serve any IoT devices in the environment while minimizing the amount of non utilized resources.

References

[1] Zhiqiang Wei, Shuwei Qin, Dongning Jia, and Yongquan Yang. Research and design of cloud architecture for smart home. In *2010 IEEE International Conference on Software Engineering and Service Sciences*, pages 86–89. IEEE, 2010.

[2] Tam Thanh Doan, Reihaneh Safavi-Naini, Shuai Li, Sepideh Avizheh, Philip WL Fong, et al. Towards a resilient smart home. In *Proceedings of the 2018 Workshop on IoT Security and Privacy*, pages 15–21. ACM, 2018.

[3] Thomas Zachariah, Noah Klugman, Bradford Campbell, Joshua Adkins, Neal Jackson, and Prabal Dutta. The internet of things has a gateway problem. In *Proceedings of the 16th international workshop on mobile computing systems and applications*, pages 27–32. ACM, 2015.

[4] Asad Javed, Keijo Heljanko, Andrea Buda, and Kary Främling. Cefiot: A fault-tolerant iot architecture for edge and cloud. In *2018 IEEE 4th World Forum on Internet of Things (WF-IoT)*, pages 813–818. IEEE, 2018.

[5] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 13–16. ACM, 2012.

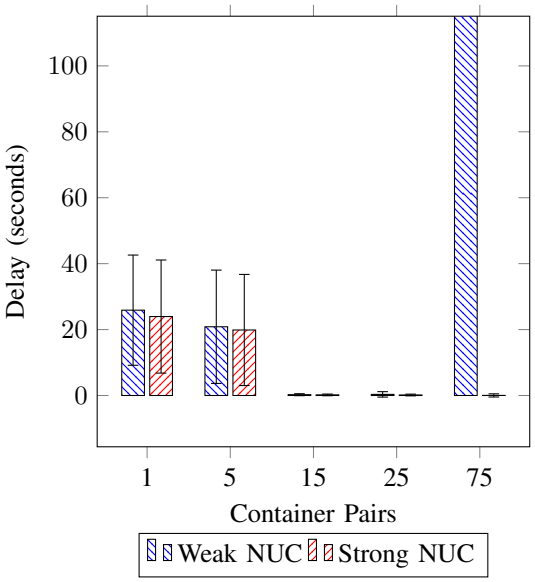
[6] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646, 2016.

[7] Shashank Shekhar, Ajay Dev Chhokra, Anirban Bhattacharjee, Guillaume Aupy, and Aniruddha Gokhale. Indices: exploiting edge resources for performance-aware cloud-hosted services. In *2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*, pages 75–80. IEEE, 2017.

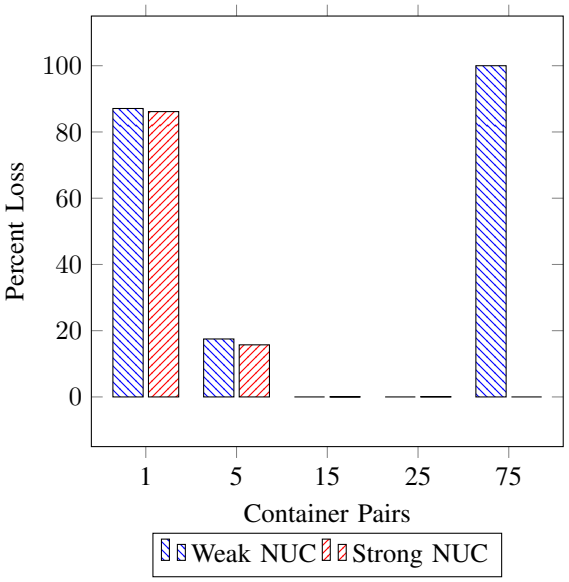
[8] Jianhua Li, Jiong Jin, Dong Yuan, Marimuthu Palaniswami, and Klaus Moessner. Ehopes: Data-centered fog platform for smart living. In *2015 International Telecommunication Networks and Applications Conference (ITNAC)*, pages 308–313. IEEE, 2015.

[9] Yehia Elkhatib, Barry Porter, Heverson B Ribeiro, Mohamed Faten Zhani, Junaid Qadir, and Etienne Rivière. On using micro-clouds to deliver the fog. *IEEE Internet Computing*, 21(2):8–15, 2017.

[10] Zodiac GX Hardware Specifications. https://northboundnetworks.com/pages/zodiac-gx-zodiac-gx-hardware-specifications. Accessed: 2020-08-18.



(a) Delays



(b) Losses

Fig. 9: Delay results, load balancing

[11] Intel® NUC Kit NUC6CAYH . https://www.intel.com/content/www/us/en/products/boards-kits/nuc/kits/nuc6cayh.html. Accessed: 2020-08-28.

[12] LG01-P IoT Gateway feat. LoRa® technology. https://www.dragino.com/products/lora/item/117-lg01-p.html. Accessed: 2020-08-18.

[13] Mikrotik hEX . https://mikrotik.com/product/RB750Gr3. Accessed: 2020-09-01.

[14] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, Bob Lantz, and Nick McKeown. Reproducible network experiments using container-based emulation. In *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, CoNEXT '12, page 253–264, New York, NY, USA, 2012. Association for Computing Machinery.

[15] SuperChassis 118G-1400B . https://www.supermicro.com/en/products/chassis/1U/118/SC118G-1400B. Accessed: 2020-08-28.

[16] Intel® NUC Kit NUC8i7HVK . https://www.intel.com/content/www/us/en/products/boards-kits/nuc/kits/nuc8i7hvk.html. Accessed: 2020-08-28.